Detecting, Validating and Characterizing Computer Infections in the Wild

Elias Raftopoulos
ETH Zurich
Zurich, Switzerland
rilias@tik.ee.ethz.ch

Xenofontas Dimitropoulos
ETH Zurich
Zurich, Switzerland
fontas@tik.ee.ethz.ch

ABSTRACT

Although network intrusion detection systems (IDSs) have been studied for several years, their operators are still overwhelmed by a large number of false-positive alerts. In this work we study the following problem: from a large archive of intrusion alerts collected in a production network, we want to detect with a small number of false positives hosts within the network that have been infected by malware. Solving this problem is essential not only for reducing the false-positive rate of IDSs, but also for labeling traces collected in the wild with information about validated security incidents. We use a 9-month long dataset of IDS alerts and we first build a novel heuristic to detect infected hosts from the on average 3 million alerts we observe per day. Our heuristic uses a statistical measure to find hosts that exhibit a repeated multi-stage malicious footprint involving specific classes of alerts. A significant part of our work is devoted to the validation of our heuristic. We conduct a complex experiment to assess the security of suspected infected systems in a production environment using data from several independent sources, including intrusion alerts, blacklists, host scanning logs, vulnerability reports, and search engine queries. We find that the false positive rate of our heuristic is 15% and analyze in-depth the root causes of the false positives. Having validated our heuristic, we apply it to our entire trace, and characterize various important properties of 9 thousand infected hosts in total. For example, we find that among the infected hosts, a small number of heavy hitters originate most outbound attacks and that future infections are more likely to occur close to already infected hosts.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: [General Security and Protection]; C.2.3 [**Network Operations**]: [Network Monitoring]

General Terms

Measurement, Security, Verification

Keywords

Network Security, Intrusion Detection, Alert Correlation, Malware, Snort, J-Measure

1. INTRODUCTION

Evaluating and improving network defenses necessitates the use of realistic traces, like IDS alerts, from production networks labeled with information about validated security incidents. Although this is a well-known and long-held problem, presently the community is largely lacking both real-world security data and systematic techniques for evaluating network defenses. Given a database of IDS alerts, it is critical to find and validate security incidents in order to build benchmarks for evaluating network defenses. Motivated by this problem, in this work we introduce a heuristic to detect and propose an approach to validate active infections in our infrastructure. An infection is simply a client or a server with malicious software, which, in our context, leaves a network trace detectable by an IDS sensor. For example, the malware could be a trojan, worm, spyware, backdoor, etc.

The second problem that motivates our work is IDS false-positive reduction in the context of extrusion detection. Modern malware increasingly involve the user in their propagation by leveraging various social engineering techniques that bypass intrusion prevention measures. For this reason, security administrators need tools for detecting hosts within their network, i.e., *extrusion detection*, that are already infected by malware. Detecting extrusions from IDS alerts bears the challenge of reducing the large number of false positives IDSs are known to generate, e.g., figures of 99% false positives have been reported in the literature [33].

Our first contribution is a heuristic for finding infected hosts from IDS alerts. Our heuristic uses an information theoretic measure, called *J-Measure*, to identify statistically significant temporal associations between a selected pool of alerts. In this manner, it finds hosts that exhibit a recurring multi-stage alert trace. To validate that hosts with this footprint are indeed infected, we conduct a complex experiment: over a period of approximately one month we remotely assess a number of live suspected infections on unmanaged hosts within a production environment. Using six independent security-related information sources, namely IDS alerts, blacklists, threat reports, host scanning logs, vulnerability reports and search engine queries, we conclude that our heuristic reduces the amount of false positives to approximately 15%. In addition, we analyze the root causes of the false positives and draw insights for filtering them out.

Then, we apply our heuristic to 832 million alerts collected over a period of 9 months and identify 12,915 different infections on 9,163 out of the of the 91,512 distinct hosts that generated IDS alerts. We characterize the extracted infections and make a number of important observations:

- Out of a total of 91 thousand distinct active hosts we observed during the 9-month period, approximately 9% exhibited signs of infections at least once during their lifetime.
- The probability of infection for a server and a client during a specific day is 0.18% and 0.37%, respectively.
- Infections drastically increase the attractiveness of infected hosts to further inbound attacks.
- A small percentage of hosts are popular sources and targets of attacks. In particular, 5% of the internal hosts account for more than 70% of the total recorded attacks originating from the intranet. In addition, servers are much more preferable targets than clients.
- Healthy hosts closer in terms of IP address distance to infected hosts are much more likely to become infected.
- The infection points exhibit diurnal patterns.
- Our time series analysis shows that server infections are almost independent in time, while client infections are consistently more bursty and this is more evident for aggregation time-scales above two minutes.

In summary, in this work we make the following **contributions**:

1. **Detection**: We introduce a heuristic for detecting infections that uses a statistical correlation measure, namely the *J-Measure*, to find hosts that produce a recurring multi-stage alert footprint involving specific classes of alerts.
2. **Validation**: We introduce a methodolgy and conduct a complex experiment to systematically validate suspected live infections on unmanaged hosts within a production environment. We find that the false positive rate of our heuristic is 15%.
3. **Characterization**: We characterize 9,163 infections over a period of 9 months. To the best of our knowledge, this is the first characterization of a very large number of infections.

The remainder of this paper is structured as follows. In Section 2 we describe the IDS alert traces we used in our experiments. We introduce our heuristic in Section 3 and describe our validation experiments and results in Section 4. Then, we characterize a number of interesting properties of the identified infections in Section 5. Finally, we review related work in Section 6, we discuss our findings in Section 7 and conclude our paper in Section 8.

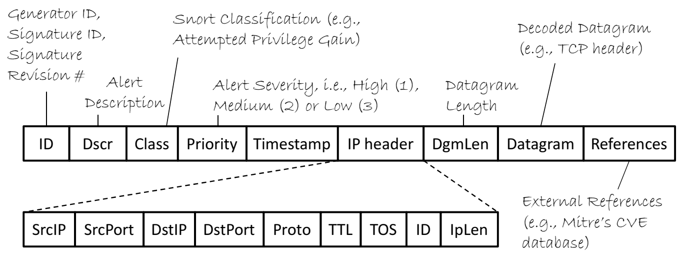
Generator ID, Signature ID, Signature Revision #
Alert Description
Snort Classification (e.g., Attempted Privilege Gain)
Alert Severity, i.e., High (1), Medium (2) or Low (3)
Datagram Length
Decoded Datagram (e.g., TCP header)
ID
Dscr
Class
Priority
Timestamp
IP header
DgmLen
Datagram
References
External References (e.g., Mitre's CVE database)
SrcIP
SrcPort
DstIP
DstPort
Proto
TTL
TOS
ID
IpLen

Figure 1: Snort Alert Full Format

2. IDS DATA

Our dataset is comprised of raw IDS alerts triggered in the main campus of ETH Zurich by a Snort [47] sensor, which is placed between the edge router of the campus and the network firewall. The sensor monitors all the upstream and downstream traffic of the campus. It uses the official Snort signature ruleset and the Emerging Threats (ET) ruleset [19], which are the two most commonly-used Snort rulesets. As of April 2011 the two rulesets have a total of 37,388 distinct signatures to detect malicious activities.

The collected alerts have the standard full Snort format shown in Figure 1. For example, the following is an actual high priority alert (with anonymized IP addresses) about a suspected MySQL bot:

```
[**] [1:2001689:7] ET WORM Potential MySQL bot scanning for SQL server [**]
[Classification: A Network Trojan was detected] [Priority: 1]
01/01-22:04:51.319793 aaa.bbb.ccc.ddd:41276 -> xxx.yyy.zzz.hhh:3306
TCP TTL:61 TOS:0x0 ID:14368 IpLen:20 DgmLen:44 DF
******S* Seq: 0xC2A22307  Ack: 0x0  Win: 0x16D0  TcpLen: 24
```

The fields we use are the unique rule identification number, the rule description, the timestamp that denotes when the alert was triggered, the IPs and ports of the communicating hosts, the default rule classification, which indicates the type of suspected malicious activity, and the rule priority, which provides a severity rank. The complete raw alerts as generated by Snort are sent every hour to our collection and archiving infrastructure.

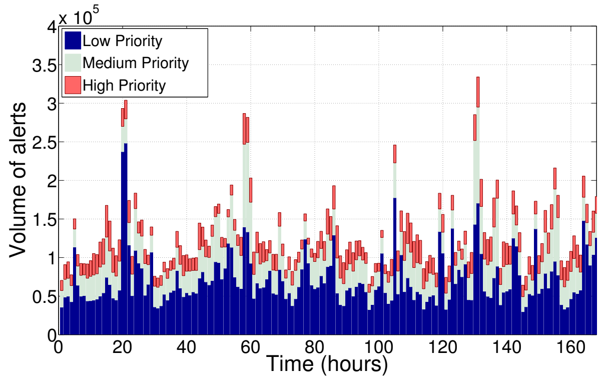

Figure 2: Volume of low, medium, and high priority alerts per hour during a period of a week

The dataset is both large and rich. During the 9 month period we study, spanning from January 1st 2010 to September 22nd 2010, our monitoring ran on a 24/7 basis with only minor interruptions (corresponding to approximately 99% availability), capturing more than 832 million alerts

Table 1: Classtype frequency of rules in `sql.rules`

#	Classification	Description
691	misc-activity	Miscellaneous activity
293	successful-recon-limited	Information leak
52	attempted-admin	Attempted administrator privilege gain
22	attempted-user	Attempted user privilege gain
4	unsuccessful-user	Unsuccessful user privilege gain
3	shellcode-detect	Executable code was detected
2	suspicious-login	An attempted login using a suspicious username was detected
2	misc-attack	Miscellaneous attack

from 91,512 thousand internal IPs. Figure 2 illustrates the amount of alerts that we collect during a regular week. On an hourly basis we record on average more than 130 thousand alerts. The vast majority of these alerts have low priority and usually correspond to policy violations that are not directly related to security incidents. However, a significant portion, approximately 6%, consists of high priority alerts.

To identify unique host infections, we restrict our analysis to hosts with static IP addresses and exclude alerts from dynamic IP address ranges. We distinguish between dynamic and static subnets using a catalog maintained by our network administrators that documents each campus subnet. Additionally, this information enables us to find whether a subnet accommodates server or client machines. The excluded alerts originating from dynamic IP address ranges, correspond to less than 17% of the total active internal IPs in our data. The fact that most hosts use static IP addresses is important as it enables us to track and characterize their behavior over time.

3. METHODOLOGY

3.1 Alert Bundling

The first challenge that we need to deal with is that security events often trigger spurts of very similar alerts. For example, certain types of port scanning targeting a range of destination ports will generate a large number of almost identical alerts that only differ in the destination port and timestamp fields. Besides, malware often change slightly their behavior in order to evade detection. Snort rulesets often include different signatures for each different malware version. When the malicious behavior is manifested, multiple versions of the same signature may be triggered in a very short time window. For example, we observe spurts of the alert ”ET DROP Known Bot C&C Server Traffic group (X)” that only differ in the version number X. Such spurts of almost identical alerts are not desirable, since they defuse a single event into multiple segments. Alert bundling groups spurts of very similar alerts into a single aggregate alert. Compared to different forms of alerts aggregation, which have been studied in the literature [51], alert bundling aims at aggregating spurts of almost identical alerts instead of creating groups of much more diverse alerts that correspond to the same aggregate multi-stage incident. Alert bundling is useful as it reduces the amount of alerts that need to be processed and facilitates the statistical analysis of different events.

We perform alert bundling over three fields, source/destination ports and alert ID. We generalize the port fields from a numerical value to {*privileged*,*ephemeral*}, based on whether the port number is below or above 1024, respectively. We also generalize alert IDs that correspond to different flavors of the same malware into a single alert ID by ignoring the version number. We then merge alerts triggered within a short time window into a single generalized alert. We preserve the timestamp of the first alert of the merged sequence. We select an aggregation window of 5 seconds. Our calibration showed that this is sufficient to substantially reduce the number of alerts, while further increasing this window had a negligible effect on the volume of alerts. Alert bundling reduced the total number of alerts in our data by 19%.

3.2 Alert Classification

Our dataset includes alerts triggered from 37,388 thousand unique rules. Snort rules are mainly community-contributed and follow a loose two-level classification scheme. Each rule is part of a ruleset, which groups related rules. For example, the ruleset `imap.rules` groups rules associated with the IMAP protocol. The second level of classification is based on the class field that is contained within each rule. The class field associates each rule with a unique class that provides information regarding the intended goal of an intrusion.

For our purposes, we find the default two-level classification scheme insufficient to extract alerts that relate to attacks and compromised hosts, which are the types of alerts we are interested in. The first shortcoming is that rules are grouped into rulesets based on different criteria. For example, some rulesets, like `imap.rules` and `voip.rules`, group rules based on the protocol or the application that is targeted, while some other rulesets, like `ddos.rules`, groups rules based on the type of the intrusion. A second problem is that rulesets often contain very diverse rules. For example `sql.rules` contains rules that range from accessing a database, which could correspond to benign behavior, to SQL worm propagation, which could indicate an infected host. Moreover, the classes associated with the classtype field are scarcely documented and in some cases ambiguous. In Table 1 we list the classes for alerts in the `sql.rules` file and provide the official documentation for each class. Some classes are quite intuitive, for example *Attempted administrator privilege gain* denotes that a privilege escalation attack took place. However, some other classes, like *Miscellaneous activity*, are quite cryptic and can result in loose classifications.

To address this problem, we use a hierarchical approach to classify the rules included in our data into three classes, namely *Attacks*, *Compromised hosts*, and *Policy violations* (similarly to [30]). In the first step, we manually examined all the rulesets and identified the ones that clearly character-

Table 2: Rulesets and classtypes assigned to the *Compromise* class

Rulesets	Description
attack-responses.rules	Privilege escalation attempts
backdoor.rules	Trojan activity operating as Backdoor
ddos.rules	Bot initiating a DDoS attack
virus.rules	Malicious code attempting to propagate
emerging-botcc.rules	Bot-related trojan activity
emerging-compromised.rules	Attacks from blacklisted IPs
emerging-user_agents.rules	Data stealing malware
emerging-virus.rules	Malicious code attempting to propagate
Classtypes	**Description**
trojan-activity	A network Trojan was detected

ize an attack or a compromised host. With this step we were able to classify 72.5% of the total number of rules. For the remaining set of rules, we used the classtype field and identified 16 classes that can be clearly associated with attacks or compromised host activity. Finally, for the remaining 681 rules, we manually classified them by examining the details of the signature, the assigned default priority level, the exact byte sequence, and when possible we validated our results with information provided in security archives and bulletins [47, 20]. In Table 2 we summarize the rulesets and classtypes we used for our *Compromise* class.

Finally, the alerts that are not classified as attacks or compromised hosts, mostly occur when a user does not comply with a specific policy. Typically these alerts correspond to P2P, VoIP, and chat related rules. We discard these rules since they do not provide any useful information about infections. For the remaining sections, we only work with alerts of the *Attack* and *Compromise* class.

3.3 Identifying Infections

A naive approach in identifying infections of internal hosts is to rely on occurrences of *Attack* and *Compromise* related alerts. However, the excessive amount of false positives, makes it very hard to have any level of confidence that we can infer an actual infection using a single alert.

We build our heuristic to extract infections based on the following design goals.

- **Keep it simple**: We opt to keep our heuristic simple as parsimony provides a number of advantages: 1) inferences are interpretable and easier to trace and validate both for a scientist and an IDS operator; and 2) the heuristic can efficiently analyze large archives of millions of IDS alerts.

- **Reduce false positives**: The number of false positives is involved in a fundamental trade-off with the sensitivity of the detector. Presently, IDSs suffer from a very large number of false positives. In this trade-off, we opt to make our heuristic conservative, i.e., less sensitive, so that the inferences it produces include a small number of false positives. This also means that we may incur some false negatives, which we prefer than triggering a large number of false positives. In order to reduce the number of false positives, we engineer our heuristic to combine multiple evidence.

- **Detect recurring multi-stage behavior**: Pre-sently, malware developers bundle a plethora of features and capabilities to make their product more attractive. For example, malware attempt to redirect users to malicious websites and download additional trojans; they update, receive instructions, share confidential data, and participate in (D)DoS attacks or spamming campaigns; they attempt to propagate by scanning for exposed nodes and by exploiting vulnerabilities, etc. This means that most modern malware exhibit a multi-stage network footprint. Additionally, the multi-stage behavior is typically recurring. For example, a host infected with an SQL worm, will scan for vulnerable machines running an unpatched version of the Microsoft SQL server. Every time a target is found, the infected host will initiate a buffer overflow attack in order to exploit the vulnerability and eventually infect the victim. A Zeus trojan will attempt to inject fake HTML code every time the user visits an online bank page, in order to steal confidential data. The collected details will be then delivered to databases residing in a remote site. Based on these observations, our heuristic attempts to reduce the number of IDS false positives by searching for malware that exhibit a recurring multistage behavior.

- **Focus on extrusion detection**: Our heuristic aims at detecting hosts within an organization that are already infected. It does not try to proactively prevent an infection.

Detection Heuristic: Our approach aims at detecting a recurring multi-stage footprint generated by infected hosts. In the simplest case, a multi-stage footprint resolves into tuples of strongly correlated alerts. Such tuples capture different actions undertaken by an infected host that occur frequently and consistently over time, increasing our certainty that an actual infection has indeed occured. We use an entropy-based information-theoretic criterion to detect significant tuples of alerts.

Our input data is a time series of alerts, where each alert is identified by the following five fields: *<ID; SrcIP; DstIP; SrcPort; DstPort>*. We examine each internal host separately, discretize its sequence of alerts into time windows of length T, and mine for tuples of the type: if alert X occurs, then alert Y occurs within the time window T. We denote the above tuple with $X \Rightarrow Y$. Each tuple is associated with a frequency and a confidence, where the frequency is the normalized number of occurrences of the first alert X and the confidence is the fraction of occurrences that alert X

is followed by alert Y within T. A well-known measure of tuple significance that combines these two basic metrics and enables to rank tuples is the *J-Measure* [46] (for an overview of tuple ranking methods refer to [40]):

$$\textit{J-Measure}(Y;X) = P(X)(P(Y|X)log(\frac{P(Y|X)}{P(Y)})+ P(\bar{Y}|X)log(\frac{P(\bar{Y}|X)}{P(\bar{Y})})),$$

where $P(X)$ is the probability that alert X occurs; $P(Y)$ is the probability of at least one Y occurring at a randomly chosen window; $P(Y|X)$ is the probability that alert X is followed by at least one alert Y within T; and $\bar{Y}$ denotes the event that Y does not occur. Intuitively, the first term $P(X)$ captures the frequency of X, while the second term is the well-known cross-entropy and captures the average mutual information between the random variables X and Y. In this way, the *J-Measure* ranks tuples in a way that balances the trade-off between frequency and confidence.

The cross-entropy between X and Y drops when the two events tend to occur together. In particular, there are two cases when the corresponding entropy of Y drops. When X happens, Y always happens, or it doesn't ever happen. Clearly, the first case is of interest to us, since it reflects the probability of the two alerts co-occurring in a specific time window T. The second case is irrelevant since there will always be numerous alerts that do not occur when a specific alert happens, resulting in an inflated *J-Measure* value. Therefore, we only keep the left term of the cross-entropy to evaluate the significance of a tuple.

One desirable characteristic of the *J-Measure* is its limiting properties. Its value ranges from 0, when random variables X and Y are independent, to $\frac{1}{P(Y)}$, when they are completely dependent, which facilitates the process of defining a threshold above which tuples are considered significant. An internal host that produces at least one significant tuple is considered infected. We fine-tune the threshold to $\frac{0.85}{P(Y)}$ as described in Section 4.4 using validated infections from our most reliable source, which is security tickets about infected and remediated systems by our security group. From the set of significant tuples we can easily extract the infection timestamps. For each tuple $X \Rightarrow Y$, if there is no other tuple $Z \Rightarrow W$ involving the same internal host within a time window T_{inf}, then this is a new infection at the timestamp of alert X. Otherwise, this is an ongoing infection and we ignore the corresponding tuple.

In Algorithm 1 we show the pseudo-code of our heuristic. Its complexity is $O(n^2)$, where n is the number of unique alerts triggered by an internal node during T. In our experiments n is quite low and on average equal to 3.1. To run our heuristic on one day of data takes on average 19.3 minutes on a system running Debian Etch with a 2GHz Quad-Core AMD Opteron.

Parameter Tuning: For the window size T we conservatively select one hour, since most alerts related to the same infection in our data occur within minutes. Selecting a larger window has negligible impact on the results. Moreover, we consider that a host is re-infected if the host is active in our dataset, but for a period of T_{inf} it is not detected as infected by our heuristic. We set the T_{inf} threshold to two weeks. We select this value in a conservative way based on two observations. Incidents identified and investigated in the past

Algorithm 1 Pseudo-code of our heuristic for detecting infections

```
Require: Set $L$ of alerts triggered by internal hosts
Ensure: Significant tuples $S_i$ for internal node $i$
  for all internal nodes $i$ do
    for all hourly timebins $T_k$ do
      for all tuples ($A_i$, $B_i$) in $L$, triggered in $T_k$, where $A_i \neq B_i$ do
        if $A_i \Rightarrow B_i$ in candidate tuple set $R_i$ then
          $R_i$.UpdateTupleStats($A_i \Rightarrow B_i$)
        else
          $R_i$.AddTuple($A_i \Rightarrow B_i$)
        end if
      end for
    end for
    for all tuples $M_i \Rightarrow N_i$ in $R_i$ do
      if J-Measure($M_i \Rightarrow N_i$) > $J_{thresh}$ then
        $S_i$.AddTuple( $M_i \Rightarrow N_i$ )
      end if
    end for
  end for
```

in our infrastructure suggest that the worst case delay required by our security group to fix a reported problem is approximately one week. This time frame covers the stages of threat identification, threat assessment, and remediation of the host either by completely removing the malware or by rebuilding the entire system. On the other hand it is known, that some malware infections stay dormant for predefined time periods or wait for an external command to trigger their behavior [4]. In this case, the host will be reported as benign by our heuristic, since no network trace of malicious activity is being generated. However, after the initial stimulus and assuming that the malicious behavior has been manifested, it is highly unlikely that the malware will fall again into idle mode for a time period longer than T_{inf} [44]. Out of the total infections we find in our characterization, 4.9% are re-infections.

4. VALIDATING INFECTIONS

In this section we present the process we follow to validate inferred infections and to assess the false positive rate of our heuristic. Remotely validating several suspected infections on unmanaged hosts within a production infrastructure is a very challenging problem that to the best of our knowledge has not been previously addressed in the literature. A first challenge is that typically no single tool or information source provides sufficient evidence that an actual security incident has occurred. A second challenge is that the types of malicious behaviors we examine are diverse, ranging from multi-stage attacks and worm propagation events to complex trojan and malware communication patterns.

Our validation follows a three step process as shown in Figure 3. Given a suspected infection, we first extract useful information from six security-related independent information sources about the infected host and the remote hosts it communicates. We refer to this information as *evidence*. A collection of evidence about suspected infections is passed in real-time (e.g., within a day of the first time an infection was detected) to a security expert. The expert correlates the expected behavior of the malware with the collected evidence. If all the evidence agree with the expected behavior, then a positive assessment is made about the suspected infection, otherwise it is concluded that the infection could not be validated, i.e., it is unknown if the suspected host is indeed infected or not. We conservatively consider the latter a false positive.

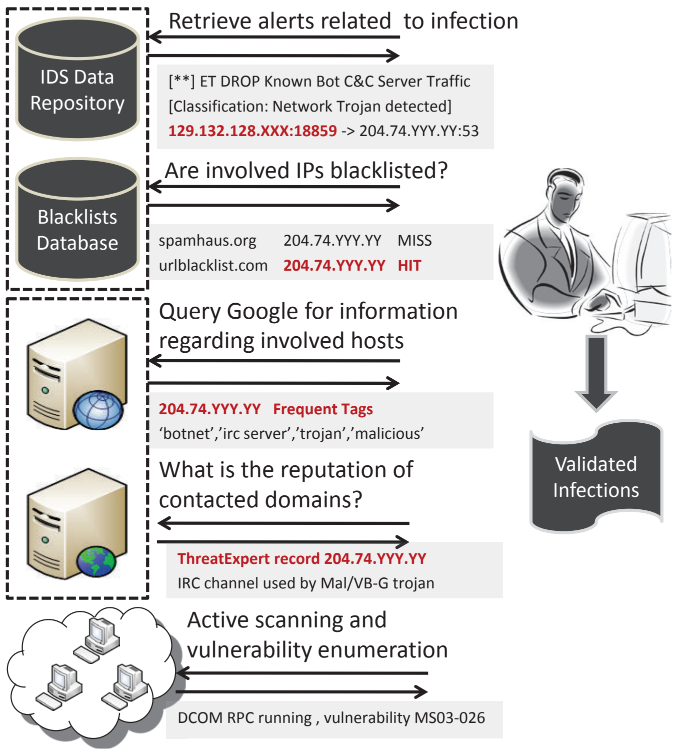
Retrieve alerts related to infection
IDS Data Repository
[**] ET DROP Known Bot C&C Server Traffic
[Classification: Network Trojan detected]
129.132.128.XXX:18859 -> 204.74.YYY.YY:53
Are involved IPs blacklisted?
Blacklists Database
spamhaus.org 204.74.YYY.YY MISS
urlblacklist.com 204.74.YYY.YY HIT
Query Google for information regarding involved hosts
204.74.YYY.YY Frequent Tags
'botnet','irc server','trojan','malicious'
What is the reputation of contacted domains?
ThreatExpert record 204.74.YYY.YY
IRC channel used by Mal/VB-G trojan
Active scanning and vulnerability enumeration
DCOM RPC running , vulnerability MS03-026
Validated Infections

Figure 3: Validation Process

This process is very demanding and time consuming for the analyst, therefore, we limit ourselves to a subset of the reported infections. Specifically, we analyze 200 consecutive incidents that were reported by our heuristic and validate the existence or absence of a variety of malware types. Although, this sample of infections is rather small compared to the total number of infections we report, the analyzed nodes are diverse spanning from servers, to desktop PCs in offices and wireless guests in labs and social areas. Also, the malicious software we investigate is quite diverse and can be categorized based on its goals and propagation methods into the following classes [37]:

- *backdoors/bots* allow an external entity to remotely control an infected machine.
- *trojans* masquerade as benign programs, but clandestinely perform illegal actions, such as information leakage and *url*-redirection.
- *worms* are self-replicating and propagating programs that attach themselves to processes or files making them carriers of a malicious behavior.
- *spyware* are useful pieces of software that are bundled with some hidden fraudulent activity.

4.1 Information Sources

IDS Alerts: For infected nodes we examine the relevant IDS alerts we have collected. We focus on alerts that are triggered in temporal proximity to the infection incident. We evaluate the quality of the alerts based on the following assumption: we consider that oversimplified signatures will tend to generate too many false positives, since it is very likely that they get triggered by benign traffic. On the other hand complex signatures are much more reliable. In order to evaluate the complexity of a signature we check if specific byte sequences within a packet are checked, if the rule specifies the ports, packet size and TCP/IP flags, if previous packets of the same flow are taken into account, and if regular expressions are being used.

Blacklists: We use independent blacklists in order to characterize a suspected host and its communicating remote hosts. We use information provided by five publicly available blacklists [22, 23, 28, 15, 17] and by one commercial blacklist [27]. We then inspect if an internal node is listed in any of these blacklists within the analyzed tracing period, and if we get a hit we tag the node based on the type of blacklist that generated the hit, e.g., spam or botnet list. Note that due to the rather high percentage of false positives [45] in most reputation-based blacklists, a blacklist hit is insufficient evidence to confirm a suspected infection. It is though a useful indicator that needs to be correlated with additional observations. Moreover, we perform the same blacklist search for external hosts that the analyzed internal machine communicated with. For example communication with hosts within the *Russian Business Network*, a network providing hosting services to cyber-criminals, could signify that the user visits some dangerous websites or that he is redirected to these URLs by an active clickbot [11] or spyware.

Threat Reports: Threat reports are publicly available security logs provided by automated systems [14] or security companies [29, 18] that analyze the behavioral patterns and common actions of a wide range of security threats including worms, trojans, and spyware. They provide a security reputation value for domains based on their observed activity during a specific interval. By investigating threat reports we can identify if a suspected host is contacting URLs that correspond to botnet rendez-vous points or malware landing pages to receive instructions, perform updates or share stolen confidential data.

Web-based Host Profiling: Apart from relying on network traces and threat analysis reports to build a security profile for a suspected host, we also use publicly available data residing on the web, which often provide useful information about the role (type of server, etc.) and involvement of hosts in security incidents [50]. This information originates from several diverse sources such as DNS-lists, website access logs, proxy logs, P2P tracker lists, forums, bulletins, banlists, IRC-lists, etc. In order to retrieve this information we query the Google search engine using as input string the IP of the analyzed host and the respective domain name we get using a reverse-DNS lookup. In an semi-automated fashion we search for tags that reveal possible roles or actions of the host such as 'trojan', 'botnet', 'spam','irc server', 'adserver', 'pop3' and 'webserver'.

Reconaissance and Vulnerability Reports: Analyzing network based data provides us with rich information regarding the behavioral patterns exhibited by a monitored host. However, we do not get any information about the running services, the patching level of critical components, and the existence or absence of vulnerabilities. Naturally, this type of information can be used to validate if an investigated node is susceptible to a specific type of infection or if the set of alerts used to infer the infection correspond to false positives, since they are not relevant to the build and operation of the specific node. Our network security assessment process consists of the following steps:

1. Host Enumeration and Basic Reconnaissance. In this step we use basic reconnaissance techniques such as IP sweeps, NIC whois querying, and TCP/UDP port-scanning in order to identify if a host is reachable and exposed to external attacks. In addition, we determine its role within the infrastructure, such as web, mail, or DNS server.

2. Network Scanning and Probing. In this step we perform targeted network scanning using nmap in order to retrieve detailed information regarding the TCP and UDP network services running on suspicious hosts, details about the OS type and version, and information regarding the types of ICMP messages a host responds to, which reveals its filtering policies and firewall effectiveness.

3. Investigation of Vulnerabilities. After having detected the accessible network services, we investigate the corresponding host for known vulnerabilities. We use publicly available sources [16, 24, 21] to identify the existence of exploitable bugs on running services. We augment this assessment with complementary information provided from vulnerability scanners, namely Nessus [25] and OpenVas [26], in order to build a comprehensive profile regarding the vulnerability status of a node.

4.2 Security Assessment

To better understand the security assessment process, in the following, we outline a set of frequent infection cases we established during our validation. For each case, we mapped the collected evidence into the behavior that was manifested by a specific malware. The four cases correspond to the four main types of malware we found in our infrastructure, namely backdoors, spyware, worms, and trojans[1].

Case 1: Backdoor infection. W32/SdBot is a typical family of IRC-controlled trojans with more than 4,000 known variants. It is used by cybercriminals as backdoor in order to gain unauthorized access to a target machine and perform unsolicited actions such as stealing private information or launching active attacks. The typical vulnerabilities we search for when we investigate an SdBot-related infection are the MS-LSASS buffer overflow, the MS-RPC malformed message buffer overflow, and the MS-WebDav vulnerabilities. These are related to the MS network shares services, which are exploited by the trojan to propagate. Regarding its command and control (C&C) activity, an infected host will attempt to use IRC to contact the adversary in order to receive instructions. This communication will typically trigger alerts with ID within the ranges [2500000:2500500] and [9000077:9000113]. The communicated C&C is typically present in our blacklst or/and profiling data. Additionally, the malware might try to propagate to other subnets. In this case we expect to see extensive scanning activity (mostly on port 445). If a vulnerable host is found and exploited successfully, then the trojan will either attempt to download a version of itself or other additional malware (typically W32/Koobface and Trojan.FakeAV) via ftp.

Case 2: Spyware Infection. The Win32/Hotbar type of malware is the most widespread infection in our infrastructure. Most variants appear as a web-browser add-on that provides a seemingly legitimate functionality. However, this malware will clandestinely steal and report user confidential data, like banking information, passwords, browsing habits, etc. For this type of infection, we find IDS alerts with IDs in the range [2003305:2003500]. We trust these alerts as the signatures are quite complex and the malware does not put any effort in disguising. Moreover, the malware operates as clickbot, changing results displayed by search engines and generating pop-ups to redirect the user to potentially malicious websites. These domains are likely to appear in our blacklists or/and web profiling data, usually with tags like 'malware-hosting', 'fraudulent', and 'phishing'.

Case 3: Worm Infection. W32/Palevo is the most common malware type found in the rather short list of worm-related infections detected in our infrastructure. It usually spreads automatically using P2P file sharing or Instant Messaging (IM) spam. When investigating this type of infection we expect to see IDS alerts with IDs in the range [2801347:2801349], which are worm specific, or more generic alerts related to activities complying with a P2P or IM protocol (typically IDs in the ranges [2451:2461], [549:565] and [2008581:2008585]). The worm will attempt to directly contact C&C nodes, without hiding the communication in an IRC channel, using an ephemeral set of port numbers. Commonly, the remote hosts `irc.ekizmedia.com`, `story.dnsentrymx.com`, and `irc.snahosting.net` are contacted. These malicious domains usually appear both in our blacklist data and in our profiling information with tags including 'botnet', 'C&C', 'Rimecud', and 'Mariposa'.

Case 4: Trojan infection. Win32/Monkif is a typical trojan that will attempt to fetch and install malicious software on a victim host. This type of malware is usually bundled with pirated software or is pushed to the victim by using phishing or social engineering attacks. When we investigate this family of infections we expect the host to connect to specific domains (including `www.clicksend.biz` and `stats.woodmedia.biz`) in order to download malicious binaries. These domains are likely to appear in our threat reports as malware hosting and generate tags as 'trojan', 'botnet', 'malware' and 'downloader' in our host profiling results.

The manual security assessment lasted for approximately one month. On a daily basis a security expert was given a list of suspected infections produced by our heuristic for the previous day along with a pool of evidence that were extracted in a semi-automated way. The expert thoroughly investigated in total 200 infections. During the first week of the validation process, two experts assessed indepedently

[1]We note that this is not a unique malware taxonomy. Putting malware into a taxonomy is challenging as most malware today have a complex behavior, usually incorporating multiple components allowing them to propagate, communicate with remote hosts to receive commands, and automatically update or initiate the download of additional malicious software.

the same suspected infections and then compared, discussed and converged on their assessments.

4.3 Validation Results

In Table 3 we summarize the number of suspected and verified infections along with the corresponding false positive rate for the four types of infections. We first note that the overall false positive rate is approximately 15%, which is remarkable. Recall that in our input data, we observe on average 3 million alerts per day, which we believe include a large number of false positives. By reversing our bundling procedure we find that only 0.6% of our input alerts of the class *Attack* and *Compromise* are associated with an infection. Our heuristic helps focus the attention of administrators to a small number of actionable cases that include substantially fewer false positives. The false positive rate for trojans, spyware, worms, and backdoors is 12.3%, 10.6%, 11%, and 35%, respectively.

Table 3: Validated infections for different infection types

	Reported Incidents	Validated Incidents	False Positive Rate (%)
Trojans	97	85	12.3
Spyware	66	59	10.6
Worms	9	8	11.0
Backdoors	28	18	35.0

Moreover, to understand better the strengths and limitations of our heuristic, we investigate the root causes of the observed false positives. The following cases were the source of most false positives.

DNS Servers. First, we find that DNS servers within our infrastructure frequently trigger signatures from the *Compromise* class. The reason is that they often attempt to resolve domains that are considered malicious. These DNS requests trigger signatures that check the destination IP address and compare it against a list of known compromised hosts. An alert will be raised in this case, typically with IDs in the range [2500000:2500941], which corresponds to backdoor related activity. DNS related false positives are mainly responsible for the inflated value regarding backdoors false positive rate shown in Table 3. However, a network administrator should be able to easily identify that these incidents do not constitute actual infections, and filter them out.

Skype Supernodes. Second, Skype supernodes within our network generate alerts with IDs in the ranges [2406000: 2406966] and [2500433:2500447]. Skype supernodes connect Skype clients by creating the Skype P2P overlay network. However, if it happens that a remote Skype user connecting to a local supernode is blacklisted, then Snort will trigger an alert identifying this system as malicious. This communication is persistent and frequent since whenever a Skype client attempts to initiate a communication, it will access a distributed database provided by supernodes in order to get the details of the contacted peer.

Antivirus. Third, a specific antivirus program generates IDS alerts of the class *Compromise* while updating. The triggered signatures check for known patterns of malicious activity found on the payload of the transmitted packets. It appears that the updates of this antivirus contain the actual pattern that it attempts to detect in plain format.

Online Games. Finally, we have observed that certain types of online games generate Snort alerts with IDs in the ranges [2003355:2003626] and [2510000:2510447]. In the case of browser-based games the triggered signatures suggest that there is an ongoing spyware-related activity. The reason is that the corresponding websites exhibit a behavior that is very similar to clickbots, attempting to redirect the player to 3rd party, potentially malicious, websites for profit. In the case of standalone gaming applications, we observe that the client will tend to preserve multiple concurrent connections with several other players. Often a small set of these remote IPs originate from domains which are blacklisted, and therefore an alert is raised.

4.4 Fine-tuning the Heuristic

As discussed in Section 3.3 an important parameter of our heuristic is the *J-Measure* threshold that determines if a specific tuple will be tagged as an active infection. In order to adjust this threshold we performed the discussed validation process on an additional small set of nodes in the local subnet of our institute. The duration of this training phase was two weeks and occurred chronologically before the main validation. During this period we run our heuristic using variable *J-Measure* threshold values and evaluated its inference results.

For the local subnet of our institute we were able to use very reliable information sources to validate a small number of detected infections. In particular, for a set of nodes we physically visited their owners and verified an infection either by performing an on the spot assessment of a system or by receiving a confirmation from an owner aware that her system was indeed infected. Secondly, our second very reliable information source for our local subnet is security tickets of our IT team. These are logged events about security incidents that have been detected, assessed, and remediated.

Using information for 28 systems we adjusted the *J-Measure* threshold in a conservative manner, i.e., aiming at keeping the false positives as low as possible, but without increasing the false negatives significantly. Selecting a threshold of $\frac{0.85}{P(Y)}$ achieved a good tradeoff, limiting the false positive rate to below 10% and the false negative rate to below 23%. For threshold values below $\frac{0.80}{P(Y)}$ the corresponding false positive rate increases above 24%, whereas for threshold values above $\frac{0.90}{P(Y)}$ we miss more than 32% of active infections.

5. CHARACTERIZING INFECTIONS

Volume and Types of Infections: The first interesting finding, illustrated in Figure 4, is that on a daily basis from an average of 11,850 active[2] hosts, we detect on average 50 new infections. The vast majority of the infected hosts correspond to client machines. Specifically 97% of the total reported incidents occur in end-host systems whereas we only see on average 10 infections per week on servers. If we normalize these numbers based on the total number of active servers and clients in our infrastructure, we see that *the probability of infection for a server system during a specific day is 0.18%, whereas the corresponding value for clients is 0.37%.*

The relatively small number of server infections can be attributed to two causes. Firstly, these systems are heavily

[2]An active host generates at least one IDS alert during an indicated period

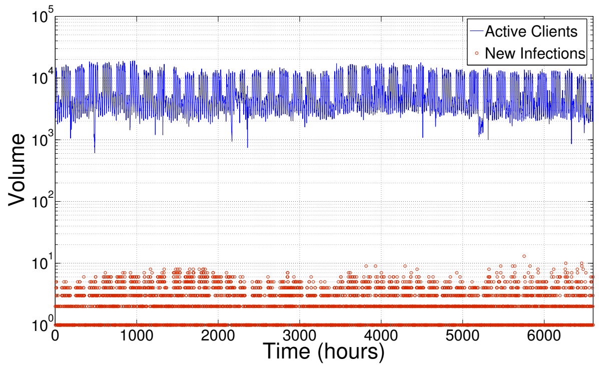

Figure 4: Active clients and new infections time-series

managed, closely monitored, and often have their OS hardened, meaning that unnecessary and potentially vulnerable services are disabled. Secondly, as we saw in Section 4 most of the malware that we observe propagate using indirect methods (e.g. drive-by-downloads, phishing) that involve the user and exploit his negligence, rather than initiating direct attacks, like scanning or buffer overflow attacks.

Moreover, *out of a total of 91 thousand distinct active hosts we observed during the 9-month period, approximately 9% exhibited signs of infections at least once during their lifetime*, whereas the total number of infections (including nodes that were re-infected) was 12,915. The number of nodes exhibiting re-infections was 675, corresponding to less than 1% of the entire active population. The majority of these nodes were connected to highly dynamic subnets in our network, corresponding to student labs and recreation areas, which are not heavily monitored. These are mostly private laptops without administrative restrictions on the installed applications and services. Therefore, the attack vector of these machines is broader, which is reflected on the increased probability of reinfection.

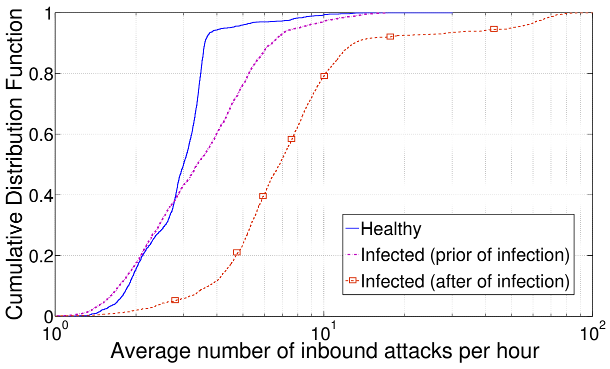

Figure 5: Infection impact on the number of inbound attacks

Infections Impact: Figure 5 illustrates the impact of an infection on the number of monitored inbound attacks. We count the average number of alerts classified as attacks targeting hosts in our intranet in an hourly basis, for all healthy hosts, for infected hosts prior to their infection, and infected hosts after their infection. Note, that based on our heuristic the infection time is estimated only after the actual infection manifests. If a node is infected but the corresponding malware remains dormant, it will not generate a malicious footprint on the network, and thus we cannot detect it. Therefore, in Figure 5, this type of nodes are considered to be in the pre-infection phase.

In the median case, healthy nodes and nodes in the pre-infection phase appear to be targets of approximately 3 attacks per hour on average. These are mostly reconnaissance attacks, such as scanning, that could be precursors of a more serious attack. The corresponding number of inbound attacks in the case of infected hosts is more than double and equal to 7. However, if we observe the tails of the distributions we see a much more sharp change. For the top 5% targets of external attacks we see that in the case of healthy nodes and nodes in the pre-infection phase we record at least 5 and 9 inbound attacks per hour on average respectively. However, in the case of infected hosts this number rises to more than 50 inbound attacks per hour.

We learn that *infections drastically increase the attractiveness of infected hosts to further inbound attacks*. We speculate that this is because most malware also operate as backdoors, allowing the installation of additional malicious code. In this way they increase the attack vector of the infected host making it a much more attractive target. This is especially true for servers, which dominate the tail of the distributions shown in Figure 5.

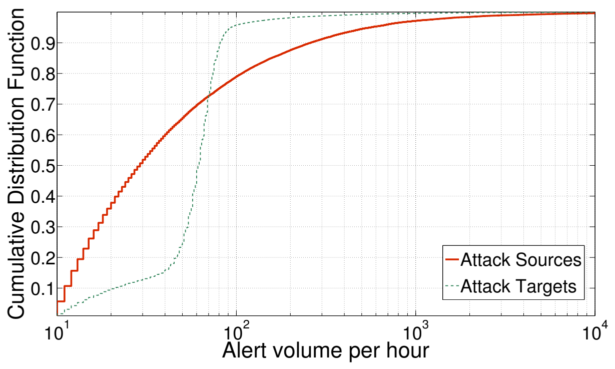

Figure 6: Distribution of alert sources and destinations for outbound traffic

Heavy Hitters: From the large pool of active clients, *a small percentage of hosts are popular sources and targets of attacks*, as shown in Figure 6. Regarding the attack popularity distribution we see that the median number of recorded inbound attacks is equal to 60 per hour. However, this number increases significantly for a small set of internal nodes that are targets to up to 970 attacks per hour. Almost, all the servers in our infrastructure are within this highly exposed set. This indicates that *servers are much more preferable targets than clients*. The reason is that most malicious pieces of self-spreading software have an initial hit-list of possibly vulnerable hosts. These hit-lists are generated using either scanning or by employing web-spiders and DNS-searches [49]. The inclusion of the public servers in our infrastructure in such hit-lists might have increased the volume of inbound attacks.

The same skewed behavior is observed in the case of the attack source distribution. We observe that approximately *5% of the internal hosts account for more than 70% of the total recorded attacks originating from the intranet*. These are highly suspicious nodes that require additional investigation. Blocking or better defending against these systems can significantly reduce the number of recorded extrusions, safeguarding at the same time exposed internal nodes. The outbound attack count could act as a crude indicator of possible malicious activity and combined with our heuristic it can facilitate administrators in focusing their efforts and identifying top offenders within their premises.

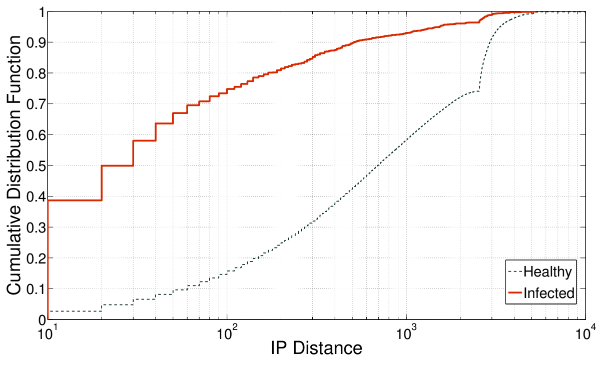

Figure 7: Infections spatial correlation

Spatial Correlations: The infections we observe exhibit strong spatial correlation. We define *IP distance* as the absolute difference between the integer representation of two IP addresses. For each host that remains healthy throughout the tracing period, we measure its *IP distance* to the closest infected host. For each host that becomes infected, we measure its IP distance to the nearest infected host at the time of infection. In Figure 7, we plot the Cumulative Distribution Function (CDF) of the IP distance for healthy and infected hosts. Note that in our infrastructure we use two large blocks of IP addresses, which explains the sharp increase we see for IP distance values above 2,600.

We observe that infected hosts are consistently in very close proximity with other infected hosts. 80% of these hosts have at least one other infected host in an *IP distance* which is less than 200, meaning that they are likely located in the same subnet. The corresponding percentage for healthy hosts considering the same *IP distance* value is significantly lower, equal to 15%.

The presence of strong spatial correlations indicates that certain subnets within a network are weak links. *Hosts close to existing infections are much more likely to become infected in the future.* Observing clusters of infections should guide administrators to review and revise the deployed baseline defenses and security policies.

Correlations Across Time: *The infection points exhibit diurnal patterns* as illustrated in Figure 8 where it is shown that most of the infections occur during working hours. This is due to the fact that client nodes, which dominate the infections set, exhibit strong diurnal patterns. This effect related to human behavior is reflected on the infections time series.

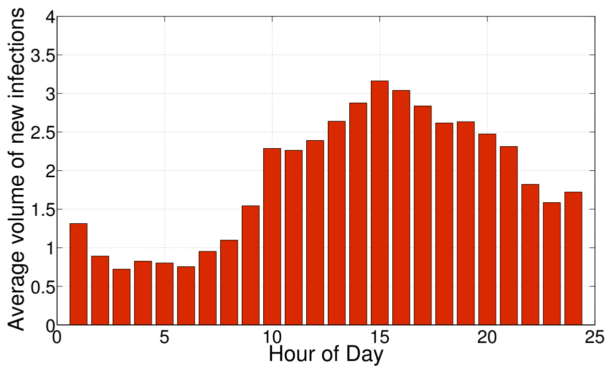

Figure 8: Distribution of infections for different hours of day

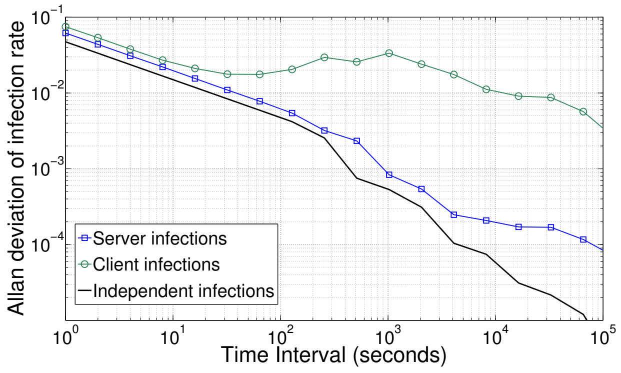

Figure 9: Infections burstiness for clients and servers

Another interesting aspect of the extracted infection time series that we study is burstiness across different time scales. To quantify burstiness, we compute the Allan deviation [2] of the infection time series at different scales. The Allan deviation is given by the following equation:

$$\sigma_x^2(\tau) = \frac{1}{2}\langle(\Delta x)^2\rangle$$

The time series is discretized into time intervals of length τ and each interval yields a sample x_i of the number of infections that occurred within it. The equation measures the difference between successive samples x_i for different interval lengths τ.

In Figure 9, the bold line in the bottom shows the minimum possible deviation which occurs when all infections have independent time arrivals. Intuitively, the Allan deviation should diverge from this reference significantly in time scales where the signal exhibits high burstiness. Figure 9 shows that *server infections in low time-scales are almost independent*, however, this changes if we look at time scales above one hour. This non-burstiness of server infections in low time scales suggests that measuring the infections over hourly intervals can provide a useful long-term average of the expected infections. This observation can be used to build a predictor of near-future infection incidents using simple linear time series models that capture short-range dependences, like ARIMA. On the other hand, *client infections are consistently more bursty and this is more evident for time-scales above two minutes.*

6. RELATED WORK

IDS Evaluation: The DARPA dataset [35] remains today one of the best options, although it dates back to 1999 and has several well-known shortcomings [1, 3]. Another well-known dataset, the DARPA Cyber Panel Correlation Technology Validation [31] was created in 2002 but unfortunately is not anymore available. These datasets were created in testbeds under controlled experiments. A lot of research has focused on generating synthetic traffic. More related to our work, MACE [48] is an environment for generating malicious packets for evaluating IDSs in testbeds. In addition to testbed experiments, in this work we stress the need to use and label and use traces collected in the wild.

Intrusion Measurements: Large traces of intrusion data, like IDS alerts and firewall logs collected by DShield [17], have been analyzed in previous studies. In particular, Yegneswaran *et al.* [52] made a number of observations regarding the distribution, types, and prevalence of intrusions. In addition, they projected the global volume of intrusions and estimated the potential of collaboration in intrusion detection. Kati *et al.* [34] analyzed a large trace of IDS alerts, reported characteristics of correlated attacks, and investigated how to effectively collaborate in intrusion detection. In this work we provide a number of further insights about intrusions focusing specifically on infections, which have not been studied as a separate class in the past.

Besides, a number of previous studies have focused on IDS alert correlation and aggregation. These studies evaluate proposed solutions on a small number of testbed-based benchmarks, like the DARPA dataset, and are tailored for general-purpose alert analysis rather than for extrusion detection. In our work, we highlight the need for using and analyzing measurements from real networks in addition to testbed-based evaluation methods. In particular, related work on alert correlation and aggregation can be classified in three categories.

Statistical/temporal Alert Correlation: A group of studies explores statistical [41, 36] or temporal [42] alert correlations to identify causality relationships between alerts. Statistical correlation methods estimate the association between different alerts by measuring the co-occurrence and frequency of alert pairs within a specific time frame. Qin [41] introduced a Bayesian network to model the causality relationship between alert pairs, while Ren *et al.* [43] proposed an online system to construct attack scenarios based on historic alert information. Temporal-based correlation approaches perform time series analysis on the alert stream to compute the dependence between different alerts. Qun and Lee [42] generate time series variables on the number of recorded alerts per time unit and use the Granger causality test to identify causal relationships. We also use a statistical alert correlation test in our heuristic and show how alert correlation can be useful specifically for extrusion detection.

Scenario- and Rule-based Alert Correlation: On the other hand, a number of studies hardcode details about attack steps into full scenarios [38, 12] or into rules [39, 6, 5], which are used to identify, summarize, and annotate alert groups. Scenario-based correlation approaches try to identify causal relationships between alerts in order to reconstruct high-level multi-stage events. Most approaches rely on attack scenarios specified by human analysts using an attack language [7, 13]. Deriving attack scenarios for all observed attacks requires significant technical expertise, prior knowledge, and time. In order to automate the scenario derivation process, machine learning techniques have been used [8]. Rule-based approaches are based on the observation that attacks can be usually subdivided into stages. These methods attempt to match specific alerts to the prerequisites and consequences of an active attack stage. The idea is to correlate alerts if the precondition of the current alert stream is satisfied by the postcondition of alerts that were analyzed earlier in time. A main limitation of scenario- and rule-based approaches is that the detection effectiveness is limited to attacks known a priori to the analyst or learned during the training phase. Therefore, they are not useful against novel attacks that have not been encountered in the past.

Alert Aggregation: The goal of alert aggregation is to group alerts into meaningful groups based on similarity criteria, making them manageable for a human analyst and amenable for subsequent statistical analysis. Each new alert is compared against all active alert sequences and is associated with the most relevant one. The set of attributes used to compute the corresponding similarity measure are diverse spanning from inherent alert features such as IP addresses, port numbers, and alert signatures, to topological and contextual information about the monitored network, such as the role and operation of nodes that triggered the alert or the deployed operating system and services. The work by Valdes *et al.* [51] represents each alert as a vector of attributes and groups alerts based on the weighted sum of the similarity of different attributes. The weight of an attribute is heuristically computed. Dain *et al.* [10, 9] propose a system that associates incoming alerts to groups in an online fashion. Julisch [32] proposes a clustering technique that aims at grouping alerts sharing the same root-causes, based on attribute-oriented induction. To address the problem that prior knowledge regarding how the alerts should be aggregated might not be available, machine learning techniques have been used. Zhu *et al.* [53] propose a supervised learning approach based on neural networks. Compared to these studies, we also use a simple form of alert aggregation in our heuristic, which we call alert bundling, that groups spurts of almost identical alerts for further statistical analysis rather than potentially diverse alerts based on complex similarity functions.

7. DISCUSSION

False Negatives: We opt to design our heuristic to produce a small number of false positives. This is one of our main goals as the excessive amount of false positives is an important limiting factor for IDSs. This means that in the trade-off between false positives and negatives we prefer to incur more false negatives in order to reduce the amount of false positives. Quantifying the false negative rate in a production environment is not possible. However, to assess false-negative rates one can use synthetic or testbed-based evaluation traces, as discussed in Section 6, where security incidents are known and controlled. Our work is complementary to such approaches and establishes techniques to find and validate security incidents in traces from production environments.

Academic Infrastructure: Our characterization results in Section 5 are based on data from an academic infrastructure and should only be carefully generalized, when possible, to other types of networks. For example, we expect that similar qualitative findings about the impact of infections and the presence of heavy hitters hold in networks of different type. In contrast, we expect that the volume of infections will be lower in more tightly managed environments.

8. CONCLUSIONS

In this paper, we present a novel approach to identify active infections in a large population of hosts, using IDS logs. We tailor our heuristic based on the observation that alerts with high mutual information are very likely to be correlated. Correlated alerts of specific types reveal that an actual infection has occurred. By applying this heuristic to a large dataset of collected alerts, we find infections for a population of more than 91 thousand unique hosts. We perform an extensive validation study in order to assess the effectiveness of our method, and show that it manages to reduce the false-positive rate of the raw IDS alerts to only 15%. Our characterization suggests that infections exhibit high spatial correlations, and that the existing infections open a wide attack vector for inbound attacks. Moreover, we investigate attack heavy hitters and show that client infections are significantly more bursty compared to server infections. We believe that our results are useful in several diverse fields, such as evaluating network defenses, extrusion detection, IDS false positive reduction, and network forensics.

9. ACKNOWLEDGEMENTS

The authors wish to thank Prof. Bernhard Plattner and Dr. Vincent Lenders for their invaluable help and fruitful discussions. Furthermore, we would like to thank Matthias Eggli for helping with the validation of the infection incidents. We would also like to thank Stephan Sheridan and Christian Hallqvist at ETH for their help in the collection and archiving of the data used in this paper. This work was supported by the Armasuisse Secmetrics [2-7841-09] project.

10. REFERENCES

[1] Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3:262–294, November 2000.

[2] D. W. Allan. Time and frequency (time domain) characterization, estimation and prediction of precision clocks and oscillators. *IEEE Trans. UFFC*, 34, November 1987.

[3] Carson Brown, Alex Cowperthwaite, Abdulrahman Hijazi, and Anil Somayaji. Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 67–73, Piscataway, NJ, USA, 2009. IEEE Press.

[4] David Brumley, Cody Hartwig, Zhenkai Liang, James Newsome, Dawn Song, and Heng Yin. Automatically identifying trigger-based behavior in malware, 2008.

[5] Steven Cheung, Ulf Lindqvist, and Martin W. Fong. Modeling multistep cyber attacks for scenario recognition, 2003.

[6] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 202–, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Frédéric Cuppens and Rodolphe Ortalo. Lambda: A language to model a database for detection of attacks. In *Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, RAID '00, pages 197–216, London, UK, 2000. Springer-Verlag.

[8] D. Curry and H. Debar. Intrusion detection message exchange format: Extensible markup language document type definition, 2003.

[9] Oliver Dain and Robert K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *In Proceedings of the 2001 ACM workshop on Data Mining for Security Applications*, pages 1–13, 2001.

[10] Oliver M. Dain and Robert K. Cunningham. Building scenarios from a heterogeneous alert stream, 2002.

[11] Neil Daswani, The Google Click Quality, Security Teams, and Google Inc. The anatomy of clickbot.a. In *In USENIX Hotbots'07*, 2007.

[12] Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, RAID '00, pages 85–103, London, UK, 2001. Springer-Verlag.

[13] Steven Eckmann, Giovanni Vigna, and Richard A. Kemmerer. Statl: An attack language for state-based intrusion detection, 2002.

[14] Advanced automated threat analysis system. `www.threatexpert.com`.

[15] Anonymous postmasters early warning system. `www.apews.org`.

[16] Common Vulnerabilities and Exposures dictionary of known information security vulnerabilities. `cve.mitre.org`.

[17] Cooperative Network Security Community - Internet Security. `www.dshield.org`.

[18] Damballa - Botnet and Advanced Malware Detection and Protection. `www.damballa.com`.

[19] Emerging Threats web page. `http://www.emergingthreats.net`.

[20] Network Security Archive. `http://www.networksecurityarchive.org`.

[21] Packet Storm Full Disclosure Information Security. `packetstormsecurity.org`.

[22] Projecthoneypot web page. `www.projecthoneypot.org`.

[23] Shadowserver Foundation web page. `www.shadowserver.org`.

[24] Symantec SecurityFocus technical community. `www.securityfocus.com`.

[25] The Nessus vulnerability scanner. `www.tenable.com/products/nessus`.

[26] The Open Vulnerability Assessment System. `www.openvas.org`.

[27] The Spamhaus Project. `www.spamhaus.org`.

[28] The Urlblacklist web page. `www.urlblacklist.org`.

[29] TrustedSource Internet Reputation System. `www.trustedsource.org`.

[30] Loic Etienne and Jean-Yves Le Boudec. Malicious traffic detection in local networks with snort. Technical report, EPFL, 2009.

[31] Joshua Haines, Dorene Kewley Ryder, Laura Tinnel, and Stephen Taylor. Validation of sensor alert correlators. *IEEE Security and Privacy*, 1:46–56, January 2003.

[32] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, 6:443–471, 2003.

[33] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375, New York, NY, USA, 2002. ACM.

[34] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating against common enemies. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, pages 34–34, Berkeley, CA, USA, 2005. USENIX Association.

[35] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 34:579–595, October 2000.

[36] Federico Maggi and Stefano Zanero. On the use of different statistical tests for alert correlation: short paper. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, RAID'07, pages 167–177, Berlin, Heidelberg, 2007. Springer-Verlag.

[37] Gary McGraw and Greg Morrisett. Attacking malicious code: A report to the infosec research council. *IEEE Softw.*, 17:33–41, September 2000.

[38] Benjamin Morin and HervÃl' Debar. Correlation of intrusion symptoms: an application of chronicles. In *RAIDâĂŹ03*, pages 94–112, 2003.

[39] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *In Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254, 2002.

[40] G. Piatetsky-Shapiro. Discovery, analysis and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. AAAI Press, 1991.

[41] Xinzhou Qin. *A probabilistic-based framework for infosec alert correlation*. PhD thesis, Atlanta, GA, USA, 2005. AAI3183248.

[42] Xinzhou Qin and Wenke Lee. Statistical causality analysis of infosec alert data. In *RAID 2003*, pages 73–93, 2003.

[43] Hanli Ren, Natalia Stakhanova, and Ali A. Ghorbani. An online adaptive approach to alert correlation. In *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment*, DIMVA'10, pages 153–172, Berlin, Heidelberg, 2010. Springer-Verlag.

[44] Vyas Sekar, Yinglian Xie, Michael K. Reiter, and Hui Zhang. Is host-based anomaly detection + temporal correlation = worm causality?, 2007.

[45] Sushant Sinha, Michael Bailey, and Farnam Jahanian. Shades of grey: On the effectiveness of reputation-based blacklists. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (MALWARE '08)*, pages 57–64, Fairfax, Virginia, USA, October 2008.

[46] P. Smyth and R. M. Goodman. An information theoretic approach to rule induction from databases. *IEEE Trans. on Knowl. and Data Eng.*, 4:301–316, August 1992.

[47] A free lightweight network intrusion detection system for UNIX and Windows. `http://www.snort.org`.

[48] Joel Sommers, Vinod Yegneswaran, and Paul Barford. A framework for malicious workload generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pages 82–87, New York, NY, USA, 2004. ACM.

[49] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association.

[50] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovi, and Antonio Nucci. Unconstrained endpoint profiling (googling the internet). In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 279–290, New York, NY, USA, 2008. ACM.

[51] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection*, pages 54–68, 2001.

[52] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: global characteristics and prevalence. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pages 138–147, New York, NY, USA, 2003. ACM.

[53] Bin Zhu and Ali A. Ghorbani. Abstract alert correlation for extracting attack strategies, 2005.

Summary Review Documentation for

"Detecting, Validating and Characterizing Computer Infections in the Wild"

Authors: E. Raftopoulos, X. Dimitropoulos

Reviewer #1

Strengths: This problem is very important and the authors present a nice solution validated using real ground truth. Well written paper with good evaluation using a lot of real data.

Weaknesses: The paper focuses on a rather narrow set of attacks. System description needs to be clearer.

Comments to Authors: The way I understood the paper is that, the J-measure heuristic first groups together alerts that are correlated somehow. Then it looks at all the alerts that are correlated for a given machine and then uses the composite signature to classify whether the infection is a backdoor infection or spyware or worm or trojan. So far it is clear, but I am not sure how the mapping is done. The whole section 4.2 gives a vague description of the different types of attacks, but does not precisely define how the matching process is done. This is a key piece missing from the paper.

Further, the system descriptions are very vague; how does one use the system in practice. What constitutes the system itself is unclear. There is a lot of manual effort, it seems, in classifying the alert groups and annotating them. There is a lot of vagueness in the description unfortunately for me to understand the exact components of the system. The output of the system is also unclear. Does it output the set of alerts that correspond to XYZ worm, XYZ trojan or it just says worm and trojan. I suspect different worms or trojans have different signatures. So, does one need to build different signatures for each of these separately in advance?

If the only output is broad classes of infection, I am not sure what the advantage of this system is compared to just combining the alerts that are for a given host. Yes, the alert group will be big, but would'nt an expert just look at the types of alerts and quickly figure out what type of an attack is ? What additional advantage one has in using the J-measure.

There is another component of the paper that involves characterizing the attacks observed in their campus trace data. This study is kind of orthogonal to the other parts of the paper. This study is also not easily generalizable to other networks. I did not know what to interpret and take out of those findings.

Reviewer #2

Strengths: Well written and evaluation; quite an important problem, relieving network administrators of the false alert overload would be quite beneficial.

Weaknesses: Its not clear to me what is automated, and what is not. In other words, how much reduction in human effort could one expect by deploying this system.

Comments to Authors: The paper argues that by grouping alerts together for a host or for some other metric, one can identify the likelihood of an infection as well as the type of infection. But how is this final step done. Does the system present these clusters to the administrator and he determines whether there is an infection? I guess I am asking how does this system exactly reduce the large fraction of useless IDS alerts. Is it mainly by grouping similar alerts into one? But if that's the case, wont a simple grouping based on some IDS alert type as well as IP be sufficient? The administrator could then quickly decide what to do from this smaller set of data.

Reviewer #3

Strengths: Important and challenging problem (reducing the false-positive rate of intrusion-detection systems). Interesting idea (to use the cross entropy of alerts to detect recurring multi-stage behavior). Impressive validation effort (to manually validate each reported infection against multiple sources).

Weaknesses: The algorithm is not well defined or explained (I *think* I got it after reading the relevant section three times). It is based on manually set parameters, and it is not clear how these should be set or how they affect accuracy (although, to be fair, I have not heard of any IDS system that does not rely on manual calibration). The evaluation of the algorithm could be (easily) improved.

Comments to Authors: The main weakness of the paper is that it does not clearly describe the proposed algorithm. The second half of Section 3.3 (which is meant to describe the algorithm) is really hard to follow. Here is my understanding of what it says: (i) the ideal goal is to detect recurring multi-stage behavior, i.e., detect sequences of alerts that occur multiple times; (ii) a simplified version of this goal is to detect sequences of 2 alerts that occur multiple times (call these "significant sequences"); (iii) to do that, the algorithm computes the cross-entropy of each sequence of 2 alerts -- low cross-entropy means that the sequence does occur; (iv) each significant sequence that is time-wise separated (by more than some threshold) from the other significant sequences is reported as an infection incident. If this is what the algorithm does, there are two ways to make it clearer: One is to explicitly state steps (i) and (ii), which are currently missing. The other is to *not* use the term "rule" to refer to sequences of alerts. When reading an IDS paper, the term "rule" makes the reader think of an IDS rule, i.e., a criterion applied to a packet to determine whether an alert should be raised. I strongly recommend using some other term -- “alert sequence” or something like that.

Another weakness is the use of manually set parameters (the J-Measure threshold, the time window, and the infection threshold). I am not sure that manual calibration is avoidable in IDS systems. However, the paper could at least provide some evidence that the proposed algorithm is not significantly sensitive to its parameters. In particular, the authors could vary the values of the parameters and show how that affects the false-positive rate of their algorithm.

In Section 4.4, it would be nice to see some discussion on the false negatives -- what kind of incidents were not detected the algorithm? In line with the previous comment, how does the false negative/positive rate of the algorithm (applied on the 28-system dataset) change as a function of the parameters of the algorithm?

Other (minor) things that I did not understand about the algorithm: What is the complexity of the algorithm? Does it compute the cross-entropy of all possible pairs of alerts within each time window? In Section 3.2, I did not get which steps of the classification are manual and which ones are automated (is the second step automated?) Also, how do the first and third steps differ? The first step “manually examined all rulesets and identified groups that clearly characterize an attack or a compromised host.” The third step “manually classified [the remaining rules] by examining the details of the signature, ...” They both sound to me like manual examination.

Reviewer #4

Strengths: There is an interesting observation on the locality of infected nodes.

Weaknesses: It is not clear what are the contributions of the paper. The methodology used is a compilation of previously proposed techniques to infer specific attacks. It is not clear how general is the proposed methodology. J-measure that is used has not been formally introduced and motivated. Despite the fact that the authors claim that their methodology is general they only address well known infections. You repeatedly mention that intrusion detection systems generate a large number of false positive rates (99%), but you do not provide any reference on this. It is not clear what is the trade-off between low rate of false positives and false negatives.

Comments to Authors: Section 3.3: The use of the J-Measure is not well motivated. J-Measure is used throughout the paper. You have to provide enough justification on why this is the right metric. As you mentioned there are other metrics have also been proposed in the literature. Thus, you have to evaluate them in your dataset.

Sections 4.2 and 4.3: Despite the fact that you claim that your framework is very general when you dive into the analysis and evaluation you attack only a few cases (some of them are well addressed in the literature). Please elaborate more on the generality and applicability of your method. You should also clearly state your contributions.

You should provide a reference that shows that off-the-shelf intrusion detection tools generate a large number of false positive rates, close to 99% as you mentioned; you have to apply these tools in your traces and then show how your proposed system reduces the false positive rate.

Section 6: The related work is too lengthy (more than two pages) and does not contribute much to the understanding of the related body of work, not it puts your contributions into context.

Your methodology identifies 9000 infected machine in the campus. It is not clear how you validated that all these machines are infected; is there a ground truth that you rely on for your statistics.

You claim that the false positive rate is low. How about the false negative -- is it high in your study? Is there a fundamental trade-off between low false positive and false negative rate?

Reviewer #5

Strengths: A working method deployed at a real-world, large-scale site. I like the balance of methodology, validation, and findings. The paper’s dataset is great and I actually find it refreshing to see work that names the actual site at which traces are collected. The paper is nicely executed and structured, and mostly well-written.

Weaknesses: The novelty of this paper is very small. IDS alert aggregation has been studied in depth in the past, and the findings in the paper are mostly confirmations of known effects. The reduction of false positives is good, but the resulting false positives of 16% are still far from great.

Comments to Authors: In the abstract you say you “assess the security of [...] live infections”. What is the security of an infection supposed to be?

The idea of malware tending to co-locate in “bad” networks is not novel, see for example Sinha & Bailey’s “Improving spam blacklisting through dynamic thresholding and speculative aggregation”.

Despite the high number of false positives it seems you make no attempts of tuning your signature set -- why? I can see practical reasons for doing so, including easy of rules updates, but unfortunately even 16% false positives, viewed in isolation, is a lousy result. I personally know of large-scale sites who use other intrusion detection approaches than Snort that fare far, far better.

I struggled with 3.3. You start out saying you want to identify real infections, and then present causality inference rules of the form “if A then B follows in time window T”, but you never say *why* you do that. I suspect you mean that the fact that two alerts (can there be more than two kinds?) occurring in sequence increases confidence in a particular attack being present. It would help to be clearer here.

I could have done without 4.2, given how tightly you’ve packed the paper. Then again, I suspect readers of different backgrounds may find it more valuable to have. 4.3, on the other hand, I found quite informative. In that section, you also say you believe your unaggregated alerts to consist virtually completely of false positives. Can’t you put a precise number to it by reversing your bundling procedure?

I can confirm your hypothesis in the “Infections Impact” section. The pay-per-install approach to malware distribution means that initial infections can trigger entire cascades of additional malware ending up on a system. However, I'm not sure I follow how this explains increase in *inbound* attacks on these systems.

It would be nice to see a "step back" section in the paper. For example, given you experience, is signature-based misuse detection the way to go? Would there be point in actively maintaining the signature set instead of accepting the crud that Snort's default ruleset with Emerging Threats spews into your logs?

Response from the Authors

We made several changes to the paper to address the comments of the reviewers. Most reviewers noted that the description of the heuristic in Section 3.3 was not very clear. A major change was that we substantially clarified Section 3.3.

Answers to specific questions (in the beginning we mark the reviewer number):
R1: Our heuristic builds on attacks detectable by snort, which provides a rather rich set of signatures for detecting attacks of various types.
R1, R2: We clarified that our heuristic does not classify the type of a detected infection; it solely detects an infection.
R2: We made more clear what it automated. The detection process of our heuristic is entirely automated. It uses an assignment of snort rules into three classes, which classification we derived manually.
R2: Trivially combining alerts from a host and passing them to an administrator would not work because it results in a prohibitively large number of suspected systems an administrator needs to manually inspect.
R3: Using a set of detection parameters is common in most detection studies. We have used security tickets for re-mediated incidents and have physically visited the owners of infected systems to derive a reliable ground truth for fine-tuning the detection threshold of our heuristic. In addition, we extended Section 4.4 to report how the number of false positives/negatives changes with the detection threshold.
R3: We discuss in detail the root-causes of the main types of false positives in Section 4.3.
R4: We provide a reference for the 99% false-positives figure in the introduction. In addition, we confirmed this figure with our data too based on the suggestion of reviewer 5.
R4: We extended Section 4.4 to report how the false-positive/negative rate changes for different thresholds.
R5: Although alert correlation has been studied extensively in the past, compared to previous studies the main novelty in our work is the characterization of 9 thousand infected hosts in a production network.
R5: The authors agree that the false positive rate could be further reduced. Some hints for improving it are given in the discussion of the root-causes of false positives in Section 4.3. This could be interesting future work.
R5: We liked the suggestion of reversing the bundling process to compute the initial false positive rate. We followed this suggestion and found 99.4% false-positive un-aggregated alerts (excluding policy alerts). This confirms our expectation.